SVL920030134US1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

INVENTOR:

Wilms et al.

TITLE:

Dynamically Capturing Data Warehouse Population Activities for Analysis, Archival, and Mining

BACKGROUND OF THE INVENTION

Field of Invention

The present invention relates generally to the field of database archival. More specifically, the present invention is related to an archiving method for extract, transform, and load tasks.

Discussion of Prior Art

As more and more daily business decisions and analyses require the use of large volumes of complex warehouse data, the time required for building warehouse data has grown exponentially. End users of a data-warehousing environment often need to complete a series of extract, transform, and load (ETL) tasks in one or more job streams within a limited time window on some periodic basis. An ETL task is defined by the manipulation of data; the extraction of data from a source, the insertion into a warehouse, or the modification of existing warehouse data. A job stream describes a set of related ETL tasks chained in a dependency tree structure. Each prerequisite task in a job stream must be completed successfully before subsequent dependent tasks can be started. If a prerequisite ETL task fails, certain actions must be immediately taken to ensure the continuation of the complete task flow, thus ensuring the execution of all subsequent, dependant ETL tasks. The continuation of task flow is necessary to ensure final warehouse data can be accurately built in a timely fashion. Capturing and managing these ETL task activities becomes vital to the delivery of current warehouse data for timely

business decision-making. Being able to meet the service level commitment in building and refreshing warehouse data is crucial to the success of a business.

Data warehouse end users need a relatively large warehouse data availability time window for complex data analysis. Therefore, warehouse administrators and operators must monitor ETL tasks performed by end users closely to ensure that these tasks are completed successfully within a designated time window. Timely and successful completion of these tasks ensures that warehouse end users can access current warehouse data with low latency. To accomplish this, exception conditions must be corrected promptly so that a scheduled ETL task flow can resume without consuming a substantial amount of end users' data analysis time.

As the volume of complex business data volume continues to grow on a daily basis and data warehouse end users continue to demand larger availability windows to access warehouse data, the need arises for a monitoring system that can provide an efficient method of problem determination and future auditing with regards to ETL tasks. To accomplish efficient problem determination and to provide an audit trail, the history of ETL task execution statuses must be preserved. However, the status of ETL tasks in a typical data-warehousing environment may not be persistent since the execution status of an ETL task changes to indicate ETL task progress; only the final execution status of a completed ETL task is stored in operational warehouse metadata. Thus, it is necessary to store operational metadata that is frequently retrieved and updated, especially during the

time of ETL task execution. When an ETL task terminates abnormally, it is necessary to have a record of all interim execution statuses for a given task prior to the failure for the purposes of problem determination and for future auditing. However, preservation of all interim execution statuses in operational warehouse metadata can potentially impact the performance of ETL tasks due to increased metadata volume. In addition, preserving all interim execution statuses creates an increase in the administrative load required to maintain and control warehouse metadata. To reduce latency for end users in a data-warehousing environment, normalized warehouse metadata must not be queried excessively while the data warehouse is online. Querying for ETL execution status, for example, reduces performance and poses the risk of misinterpreting or corrupting changing operational metadata.

In one approach, an archived warehouse metadata is used to capture and store the changes in operational warehouse metadata. This approach is limited in that archived warehouse metadata has the potential to grow without bound, thus furthering maintenance concerns. If archived warehouse metadata is "pruned" by data warehouse administrators as it grows, no audit trail records will remain for future reference. In addition, a trigger mechanism capturing the changes in operational warehouse metadata continuously providing changes to archived warehouse metadata may occur simultaneously with a data warehouse end user performing data analysis on the same archived warehouse metadata. Simultaneously accessing or operating on archived warehouse metadata can influence or even corrupt analyses.

Whatever the precise merits, features, and advantages of the above cited references, none of them achieves or fulfills the purposes of the present invention.

Thus, there is a need in the art for a dynamic method of data capture and storage of past and current statuses of ETL tasks that does not impact the performance of the existing data warehousing environment and access operations. A provision must be made for data warehouse administrators to control the time, frequency, and granularity at which changed operational warehouse metadata to be captured for analysis is stored and refreshed as well as for the capability of easily recovering damaged archived warehouse metadata. Also necessary in the art are methods for capturing ETL exception conditions and for generating error recoveries for handling exception conditions.

SUMMARY OF THE INVENTION

The system and method of the present invention utilizes operational metadata obtained from a data-warehousing environment to dynamically capture data warehouse population activities. Operational metadata containing ETL task definitions, control flow, execution schedules, and execution statuses obtained from a data warehouse is utilized by trigger mechanisms, staging tables, and an archived warehouse metadata table to store specified ETL task information. ETL information is extracted from operational metadata, filtered for specified ETL task data, transformed to a format necessary for storage in an ETL task staging table, and finally, archived in an archived warehouse metadata table.

An administrator of a data-warehousing environment specifies ETL information to monitor and capture. Thus, ETL task information is extracted from operational metadata and transformed when an update or delete trigger mechanism activates a change in operational metadata. Administrator-specified ETL task information is then stored in a staging table. At specified intervals, a staging table is refreshed with changes, or delta values, in operational data for each of the administrator-specified monitored and captured ETL tasks. Data that is overwritten, or outdated ETL task information, is then moved and stored in an archived warehouse metadata table.

In one embodiment, refresh operations on a staging table are scheduled by a system or user such that move operations performed by a delete trigger attached to a

staging table occur at a pre-determined time or frequency. ETL task data in staging table is moved rather than copied to an archived warehouse metadata table thus ensuring each entry in a staging table is processed only once. An archived warehouse metadata table is periodically refreshed with changes in operational metadata that have been stored in a staging table. An archived warehouse metadata table is queried to report completed tasks, pending tasks, duration of step execution, error codes and messages, scheduling problems and changes, and any overdue ETL task run schedules or misses; for further mining and analysis, archived warehouse metadata table is backed up before its contents are filtered and stored into lower-level tables indicating ETL task errors, completed tasks, task temporary status, and task scheduled. Lower-level ETL task tables are queried to generate reports at a more granular level, thus allowing monitoring and analyses of administrator-specified ETL tasks.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a data-warehousing environment and associated data.

Figure 2 is a state transition diagram between ETL task execution statuses.

Figure 3 is a process flow diagram of the method of the present invention.

Figure 4 is a process flow diagram for refreshing archives and backing up archived information.

Figure 5 illustrates reports generated from ETL task complete table.

Figure 6 illustrates reports generated from ETL task errors table, ETL tasks scheduled table, and ETL task temporary status table.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

While this invention is illustrated and described in a preferred embodiment, the invention may be produced in many different configurations. There is depicted in the drawings, and will herein be described in detail, a preferred embodiment of the invention, with the understanding that the present disclosure is to be considered as an exemplification of the principles of the invention and the associated functional specifications for its construction and is not intended to limit the invention to the embodiment illustrated. Those skilled in the art will envision many other possible variations within the scope of the present invention.

Figure 1 illustrates a general system diagram of the present invention comprised of operational metadata **100**, trigger mechanisms **110**, **112**, **114**, staging table **116**, and archived warehouse metadata table **118** in data-warehousing environment **120**. Operational metadata **100** is comprised of ETL information **102**. ETL information **102** further comprises ETL task execution statuses **104**, run number **106**, definitions **108**, control flow, and execution schedules. An administrator of data-warehousing environment **120** specifies ETL information **102** to monitor and capture. Subsequently, ETL task information **102** is extracted from operational metadata **100** and transformed when either of update trigger **110** or delete trigger **112** attached to operational metadata **100** are activated. Administrator-specified ETL task information **102** is then stored in staging table **116**. Information in staging table **116** is refreshed with changes in operational metadata **100** for each of the administrator-

specified monitored and captured ETL tasks by triggers **110** and **112**. As a result of pruning of staging table **116**, staging table delete trigger **114** is activated and archived warehouse metadata table **118** is refreshed. Archived warehouse metadata table **118** is static and is refreshed at specified time intervals in accordance with an administrator-specified pruning schedule. Pruning of staging table **116** and subsequent refreshing of archived warehouse metadata table **118** is performed in order to free buffer space in staging table **116** in a controlled manner.

To analyze the progress of a particular ETL task flow or job stream, an administrator of data-warehousing environment specifies ETL task status as information to be monitored and captured. Shown in figure 2 are states that an ETL task status can take on as it is being executed. Transitions between states are shown as occurring as a result of specified events, known as steps. Tracking of the current state of an ETL task during execution is necessary to determine if an ETL task has completed successfully and also determine whether error recovery measures must be deployed. Also shown in figure 2 is the order of change in execution statuses of an ETL task along with the volume and complexity of state transitions in a typical data-warehousing environment. Note that states **200 - 220** indicate the range of statuses an ETL task may take on, and transitions **222 - 260** indicate events necessary to move from a previous state to a subsequent state.

Since an ETL task may be run once or multiple times in each batch window or each processing cycle, data-warehousing environment must be able to uniquely

identify execution statuses of each ETL task; a data-warehousing environment commonly assigns a unique number to represent the run edition of each ETL task. That is, if an ETL task is run twice, the first run is given one edition number and the second run of the same task is given another edition number. Edition numbers given to an ETL task are unique. This provision is of interest with regards to batch jobs run over a database. Furthermore, warehouse data generated by each run can be represented by a unique edition number to identify multiple versions of warehouse data in the same data warehouse.

Shown in figure 3 is operational metadata **300** containing information pertinent to a specified ETL task or tasks, along with associated triggers **302, 304**. Update trigger **302** and delete trigger **304** attached to data-warehousing environment ensures continuous refreshing of archived metadata for analysis, processing, and auditing without impacting the performance of existing data warehouse processing operations. In an exemplary database management system, changes to warehouse data are logged before they are written to disk. Logged change data is used to initiate transaction rollbacks and recovery processes. A database trigger takes advantage of this built-in database mechanism without adding extensive overhead or impacting the performance.

Trigger mechanisms **302, 304** in data-warehousing environment **118** automatically detect and capture changes in the status of a specified ETL task. The attachment of update trigger **302** to operational metadata **300** automates the

population of changes in ETL task information from operational metadata **300** to staging ETL task status table **306**. The attachment of delete trigger **304** to operational metadata **300** automates the deletion of ETL task information from operational metadata **300** in the event an ETL task has been purged.

If update trigger **302** attached to operational metadata **300** extracts a completed status of failure or warning for a specified ETL task or tasks; error messages associated with the failure of specified ETL task or tasks are also extracted and transformed into a format necessary for storage in staging ETL task errors table **312**. If update trigger extracts a completed ETL task (e.g., successful, warning, failed) from operational metadata **300**; the status is extracted and transformed into a format necessary for storage in staging ETL task status table **306**. If an ETL task is purged, delete trigger **302** attached to operational metadata **300** is activated, and purged ETL task information is written to staging ETL task status table **306**. Staging ETL task status table **306** is managed by the activation of delete trigger **308**, which is activated in accordance with an administrator-specified schedule. When delete trigger **308** is activated, then archived ETL task status **310** is emptied into a backup medium or media **314** or equivalent thereof, and data deleted from staging ETL task status table **306** is moved into emptied archived ETL task status table **310**.

Prior to storage in archived ETL task status table **310**, staging ETL task status table **306** buffers incoming changes in operational metadata **300**. Because staging ETL task status table **306** stores with changes in operational metadata **300**, periodic

pruning is necessary. Periodic pruning of staging ETL task status table **306** ensures that its size does not grow without bound. Because archived ETL task status table **310** is static, it is backed up at administrator-specified periodic intervals so that it too does not grow without bound. Pruned data from staging ETL task status table **306** is then moved to archived ETL task status table **310** for analysis based on administrator-specified configurations. In one embodiment, staging ETL task status table **306** is initialized with operational metadata from data-warehousing environment **100** for selected ETL tasks. If staging ETL task status table **310** is not initialized, it starts as an empty table; otherwise, it stores current ETL task information delta values contained in operational metadata **300**.

Referring now to figure 4, delete trigger **308** is activated in staging ETL task status table **306** in step **400**. In one embodiment, refresh operations on a staging ETL task status table **306** are scheduled such that move operations performed by delete trigger **308** occur at a pre-determined time or frequency. Archived ETL task status table **310** is initialized with a full copy of the operational metadata **300** for a selected ETL task or tasks. A full copy of operational metadata **300** is selected and captured in order to establish a baseline for past data warehouse population activities. In step **402**, information in staging ETL task status table **306** is moved rather than copied to archived ETL task status table **310** to ensure that each entry in staging ETL task status table **306** is processed only once. Specified ETL task information is selected from archived ETL task status table **310** for backup, in step **404**. Also in step **404**, selected

ETL task information from archived ETL task status table **310** is then backed up onto disk, optical media, etc. In step **406**, ETL task status for selected ETL task information is determined. If the status determined in step **406** is that of failure, then ETL task errors are stored in ETL task error table in step **408**.

If it is determined in step **406** that ETL task status is not that of failure, then it is determined whether completed ETL task status, temporary ETL task status, or scheduled ETL tasks has or have been selected from archived ETL task status table **310**. In steps **410**, **412**, and **414**, selected ETL task information is extracted from archived ETL task status table **310** and stored in a corresponding ETL task table.

A system administrator or user of data-warehousing environment **118** configures the transfer of information between a staging area and an archive. In one embodiment, the granularity of data moved from staging ETL task table **306** to archived ETL task status table **310** is variable. For example, upon completion of a batch job or when problem determination is required, all ETL task information pertaining to the batch job or problem is moved together from a staging area to an archive, as opposed to separately. In another embodiment, a system administrator or user of data-warehousing environment **118** backup archived ETL task status table **310** based on the timestamp of the most recently captured entry to a file or an external storage device for future auditing.

Archived ETL task status table **310** is also queried to report completed tasks, pending tasks, duration of step execution, error codes and messages, scheduling

problems and changes, and any overdue ETL task run schedules or misses after being refreshed with data from staging ETL task status table **306**. Referring now to figure 5, ETL task completed table **500** is further queried to generate reports indicating the sequence of ETL tasks executed in a process **502**, last ETL task executed **504**, ETL task or tasks failed **506**, duration of execution of ETL tasks in process **508**, statistics associated with an ETL task run or runs **510**, and ETL task or tasks retried **512**. This provision is of interest because a completed task may be specified by a warning or failure status, in addition to a successful status. A completed ETL task specified by a warning or failure status requires the analysis of interim task execution statuses and associated ETL task information.

Shown in figure 6, are reports generated from ETL task errors table **600**, ETL tasks scheduled table **602**, and ETL task temporary status table **604**. Generated from ETL task errors table **600** are reports indicating errors associated with ETL tasks having a failed status **606** as well as ETL task IDs corresponding to those tasks finishing with a specific error **608**. Generated from ETL task scheduled table **602** are reports indicating the sequence of ETL task IDs executed in a process **610** as well as de-scheduled ETL tasks **612**. Generated from ETL task temporary status table **604** are reports indicating ETL task IDs having a specific temporary status at a specified point in time **614**.

If archived ETL task status table is queried directly, additional reports to aid in monitoring and auditing ETL tasks are generated. Examples of queries that are

performed against the accumulated ETL task execution statuses in archived warehouse metadata which is not easily, if at all, discernable from a query of operational metadata comprise: ETL task steps failed since a specified date, steps failed during last ETL task execution, errors associated with failed steps, steps completed with a specified return code, suggested action or actions to resolve unsuccessful ETL task execution, reasons behind the inability of a step to execute at a specified time, length of time necessary for a step to complete each run, statistics relevant to the execution of steps in a specific ETL task, steps requiring a retry and number of retries necessary to achieve status at completion, de-scheduled, re-scheduled, and overwritten steps, steps having a specified status (e.g., purged, canceled, removed) at any point in time, and reasons for the inability of a step to be cascaded in a defined job stream.

In another embodiment, recovery actions are customized and run for known error conditions. An administrator of data-warehousing environment is alerted to follow previously customized recovery and retry actions for failed tasks generating known errors and warnings. For example, if an ETL task attempting to connect to a database fails in its execution because of a login password changed by a system administrator, an administrator follows a previously customized procedure to update the login password and rerun the failed ETL task. For new errors and warnings, an administrator of data-warehousing environment is paged to respond to exception conditions by utilizing historical execution statuses. Archived warehouse metadata is pruned and saved by warehouse administrators at configured intervals along with a current timestamp to

associate with a backup copy; a warehouse administrator can easily identify the most recent tasks recorded in the backup for future restoration.

Additionally, the present invention provides for an article of manufacture comprising computer readable program code contained within implementing one or more modules to monitor task execution statuses during data warehouse population. Furthermore, the present invention includes a computer program code-based product, which is a storage medium having program code stored therein which can be used to instruct a computer to perform any of the methods associated with the present invention. The computer storage medium includes any of, but is not limited to, the following: CD-ROM, DVD, magnetic tape, optical disc, hard drive, floppy disk, ferroelectric memory, flash memory, ferromagnetic memory, optical storage, charge coupled devices, magnetic or optical cards, smart cards, EEPROM, EPROM, RAM, ROM, DRAM, SRAM, SDRAM, or any other appropriate static or dynamic memory or data storage devices.

Implemented in computer program code based products are software modules for: (a) tracking the state of a single or multiple ETL tasks as they are run; (b) uniquely identifying the execution status of each singly or multiply executed ETL task; (c) dynamically capturing data warehouse population activities; (d) archiving interim and final ETL task information; and (e) generating and reporting ETL task-relevant information.

CONCLUSION

A system and method has been shown in the above embodiments for the effective implementation of the dynamic capture of data warehouse population activities for analysis, archival, and datamining. While various preferred embodiments have been shown and described, it will be understood that there is no intent to limit the invention by such disclosure, but rather, it is intended to cover all modifications falling within the spirit and scope of the invention, as defined in the appended claims. For example, the present invention should not be limited by software/program or computing environment.

The above enhancements are implemented in various computing environments. For example, the present invention may be implemented on a conventional IBM PC or equivalent. All programming and data related thereto are stored in computer memory, static or dynamic, and may be retrieved by the user in any of: conventional computer storage, display (i.e., CRT) and/or hardcopy (i.e., printed) formats. The programming of the present invention may be implemented by one of skill in the art of database programming.